unfolding Color Code の誤り耐性　その 2

前回に引き続き、Color Code の unfolding 操作について誤り耐性を調べてみた。特に unfolding する際の折り目の部分のエラー検出に関して詳しく解析した。

1.　折り目付近の誤り耐性

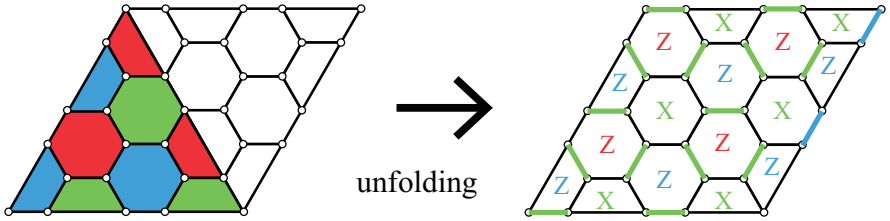

Fig. 1

前回までの unfolding 操作を Fig.1 に示す。前回の資料 (unfolding_color_code_2.pdf) ではあまり折り目の部分について議論しなかったが、よくよく考えてみると折り目の部分の誤り検出は思った以上に難しい。ここではそれを示す。

まず、最初 Color Code から始まったとき、右上部分には $|0\rangle$ 初期化されている qubit が用意してある。つまり、それらの qubit は 1-weight Z stabilizer でスタビライズされている。そのことを、水色を Z stabilizer として Fig.2(a) に示す。このとき、折り目だけに注目すると、Fig.2(b) に示すように、red face Z stabilizer と付近の 1-weight Z stabilizer で六角形の Z stabilizer が構成できる。これより、unfolding 操作の折り目をまたがる 6-weight Z stabilizer によって折り目付近の X エラーを検出できる。

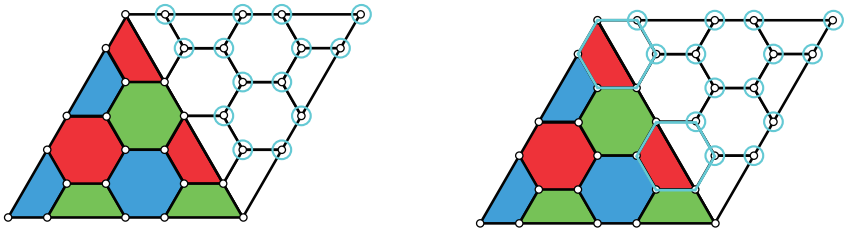

Fig. 2

しかし、Fig.3 に示すような破線の 6-weight Z stabilizer は undeterministic である。そのため、Fig.2(b) の上の水色の六角形の Z stabilizer で検出されたエラーは Fig.3 の qubit 1,2,3,4 のどこで起きたのかが全くわからない。そのようなことから、前回提示した unfolding のプロトコルは折り目付近の X エラーが取り除けない。

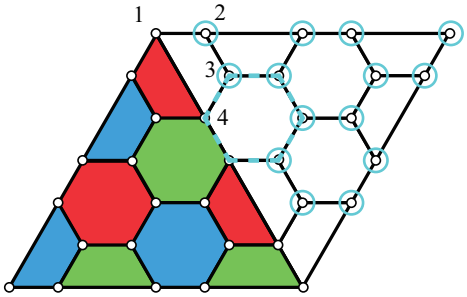

Fig. 3

また、前回のプロトコルだと折り目付近の Z エラーも取り除けないことに気がつく。しかしこれは、折り目付近の 2 つの qubit を Bell 状態にすることによって解決する。それを紫色を X stabilizer として Fig.4 に示す。ただし、このようにしても上記の X エラーを取り除けない問題は残る。

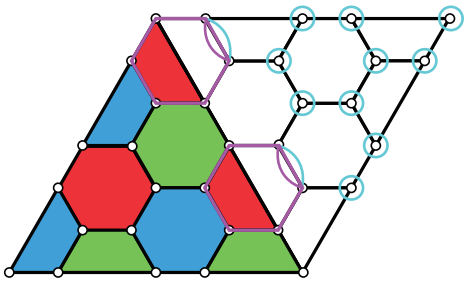

Fig. 4

ということで現状のプロトコルは修正が必要である。参考までに Fig.5 に試みた構造の残骸を載せておく。Fig.5 に示すどの構造（stabililzer 自体が成り立っていないものもある）をとっても上記と似たような問題が発生した。

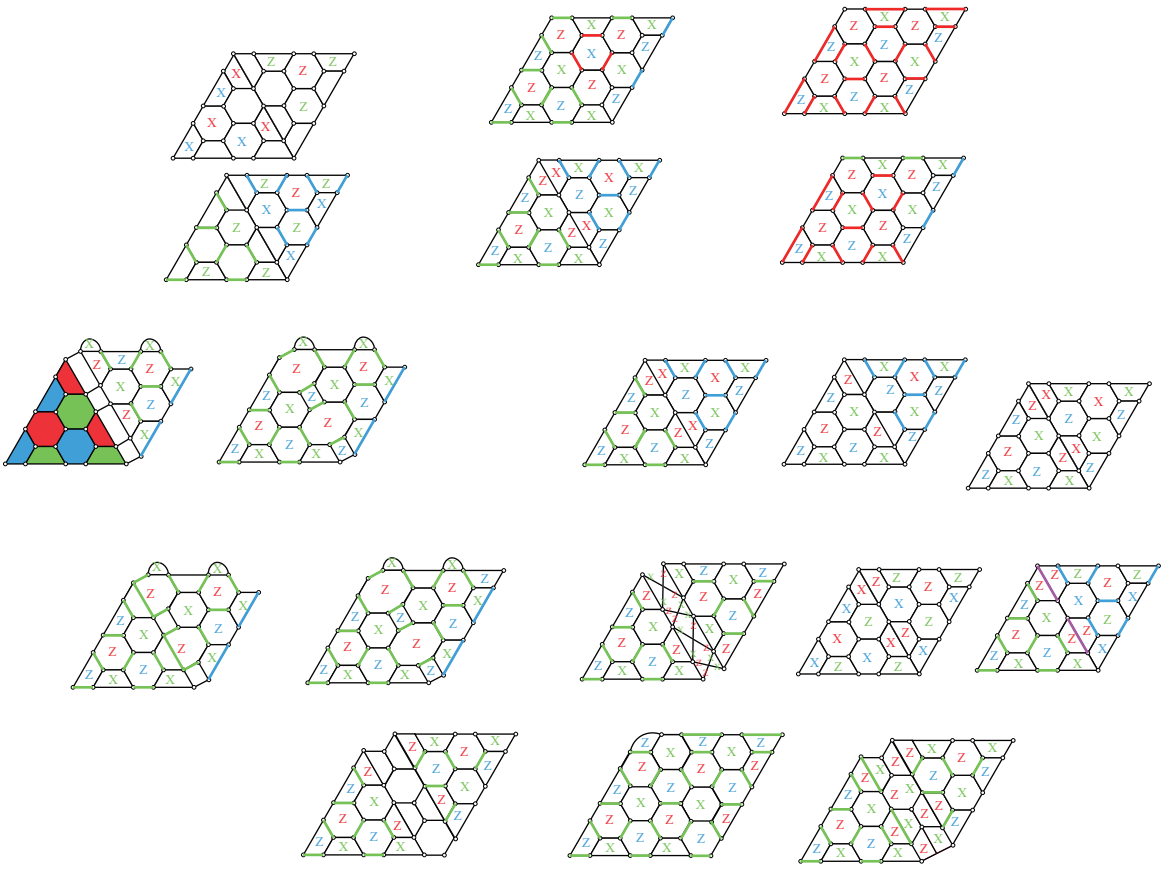

Fig. 5

ということで上記の問題の解決策になりそうなものを思いついたのでそれを以下で述べる。

2. Floquet Code×Unfolding

Floquet Code を用いるやり方を思いついたので、まずトーラス上での Floquet Code を使った Color Code の unfolding を考える。測定の種類や順番は Fig.6 に示す通りである。ここで、Fig.6 にはシンドローム測定するスタビライザーしか示していない。実際には Fig.6 はもっと多くの” 示していないスタビライザー” が存在する。

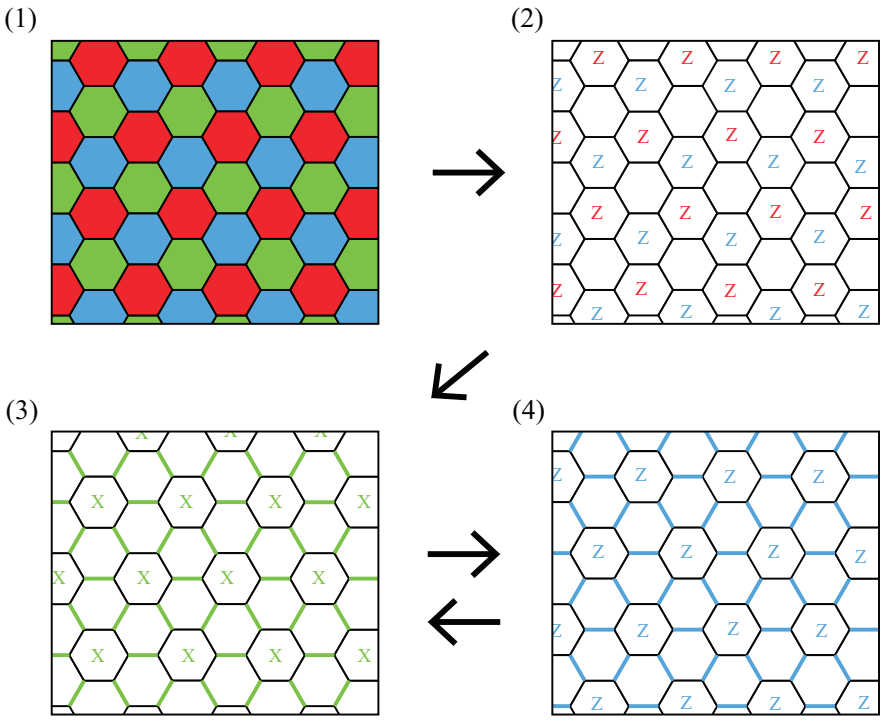

Fig. 6

ここから先、(1),(2),(3),(4) は Fig.6 の (1),(2),(3),(4) を表す。まず (1) について、最初は Color Code なのでそれぞれの face に X と Z 両方の 6-weight stabilizer がある。Z logical operator を green edge 上に 2 つ (トーラスの穴をくぐり抜けるものと、穴の周りを一周するもの) 取り、X logical operator を blue edge 上に 2 つ (トーラスの穴をくぐり抜けるものと、穴の周りを一周するもの) 取ることとする。その次に (2) で red face と blue face について 6-weight Z stabilizer をシンドローム測定すると、Color Code のときのシンドローム値と比べることによって、red face と blue face に挟まれる edge、つまり green edge 上の X エラーを検出できる。このとき、green edge 上の 2 つの qubit のうちどちらに X エラーが発生したのかわからないが、どちらかに X ゲートをかけて訂正したことにしておく。そして (3) で green face と green edge 上のスタビライザーでシンドローム測定する。このとき、一個前の (2) で訂正操作が間違っており、X エラーが発生していない方の qubit に X ゲートをかけていた場合の 2 qubit エラーは green edge 上の stabilizer を (3) で導入したことによってスタビライザーそのものとなり、訂正される。また、green face と red face 上の 6-weight X stabilizer のシンドローム値 (前回の資料と同じように red face については green edge 上のスタビライザーをかけ合わせる) がわかるので、blue edge 上の Z エラーが検出される。これも (2) のときと同じように blue edge 上の 2 つの qubit のうちどちらに Z エラーが発生したのかわからないが、(2) と同じような方法で訂正したこととしておき、(4) で blue face と blue edge 上のスタビライザーを導入することによって、Z エラーは訂正できる。また、(4) での測定によって、(2) と同じ red face と blue face 上の Z stabilizer のシンドローム値がわかり、その時に検出された X エラーは (2) のときと同じように訂正する。以後これを繰り返しエラーを訂正していく。最終的に到達する Code は Surface Code そのものである。この一連の操作で green edge と blue edge 上のスタビライザーが stabilizer generator に加えられたり、取り除かれたりされているが、stabilizer generator の総数は多分変わっていない (簡単に確認しただけで検証はしていない)。また、最初に設定した logical operator とはすべ

て可換な操作なので論理情報は保存されている。

3. Floquet Code を用いた平面上での unfolding

ここでは、section 2 で説明したことを平面で考える。測定の種類や順番は Fig.7 に示す通りである。ここで、Fig.7 にはシンドローム測定するスタビライザーしか示していない。実際には Fig.7 はもっと多くの” 示していないスタビライザー” が存在する。また、前回の資料 (unfolding_color_code_2.pdf) とは色の配置が変わっている。

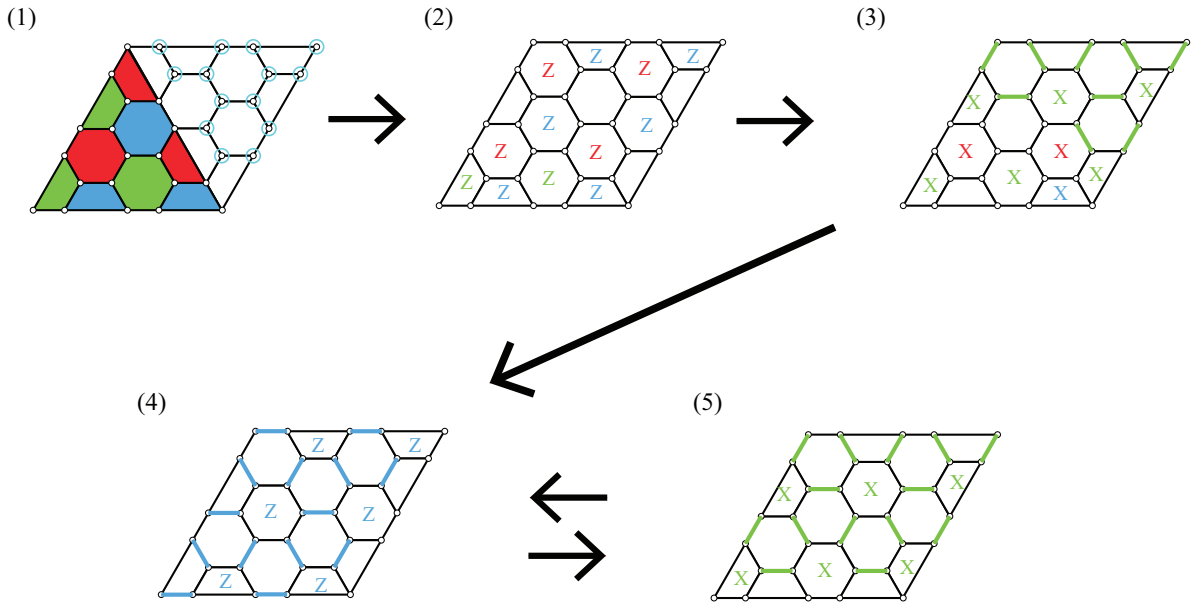

Fig. 7

ここから先、(1),(2),(3),(4),(5) は Fig.7 の (1),(2),(3),(4),(5) を表す。具体的な操作やエラー検出、訂正方法はトーラス上で行ったときと同じである。ここでは異なる部分についてのみ説明する。まず、(1) の段階で、Z logical operator を blue boudary に取り、X logical operator を red boudary に取る。そして、$|0\rangle$ 初期化されている qubit が右上に用意してある。この段階で、右上で初期化されている領域にも red face、blue face に対応する Z stabilizer が存在する。次に (2) でシンドローム測定することによって green edge 上の X エラーは (1) のシンドローム値と比較する事により、拡張した領域も含めて、検出できる。green face 上で Z stabilizer を測定している理由は Fig.8 に示す通りである。

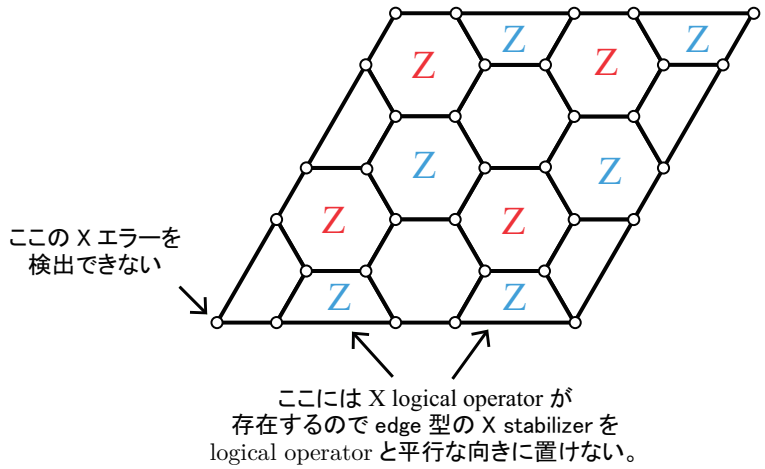

Fig. 8

green face 上の Z stabilizer が無いと、まず一番左下の qubit の X エラーが見つけられない、また、一番下の blue face Z stabilizer によって検出される boudary 上の edge(つまり、六角形を半分に切ったときにできる edge) の X エラーは edge 上の 2 qubit のうちどっちに起こったのかが完全にわからないといけない。なぜなら、その boundary 上には X logical operator が存在し、次のステップでその logical operator と平行な方向の edge には edge 型の stabilizer は置けないからである (logical operator が縮むから)。そして次に (3) について、右上部分は最初は $|0\rangle$ で初期化されているので Z エラーが存在しない。また、blue face X stabilizer が存在するがその理由は (2) の一番左下の qubit に対する議論と同じである。しかし red face の X stabilizer が存在する理由は (2) とは少し違い、Fig.9 に示す green face 上の Z stabilizer のシンドローム値を変えたくないからである。幸いなことに一部の左下の green edge X stabilizer がなくても、そもそも (2) でそこに存在する X エラーは取り除かれているので問題ない。

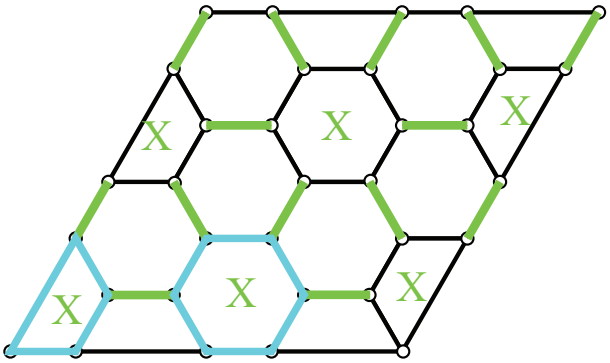

Fig. 9

そして、(4) では、すべての blue face, red face Z stabilizer のシンドローム値と (3) で守った一部の green face の Z stabilizer のシンドローム値が確定しているのですべての X エラーを取り除ける。そして (5) では全ての green face,red face X stabilizer のシンドローム値は確定しているのですべての Z エラーが取り除ける。周期 1 回目が終わったあとは平行四辺形の下と右にそれぞれ edge 型の Z stabilizer と X stabilizer が常に存在するようになるので、下と右の boundary 上は edge stabilizer のシンドローム値をそのまま用いてエラー検出を行う。

ということで最終的なスタビライザーをすべて示した図を Fig.10 に示しておく。

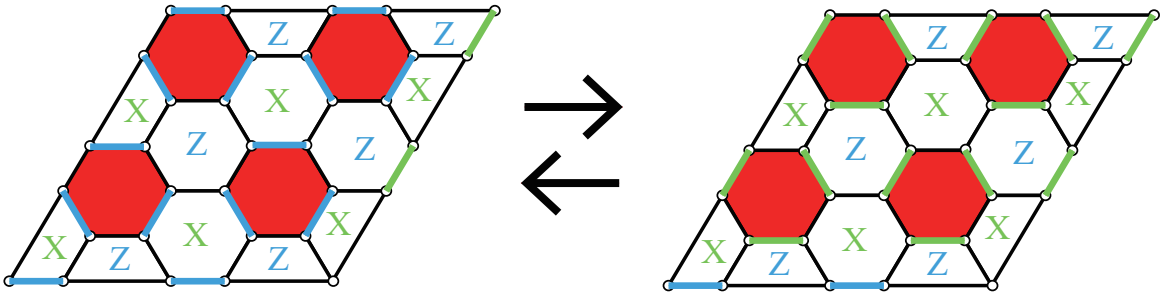

Fig. 10

4. CNOT の順序

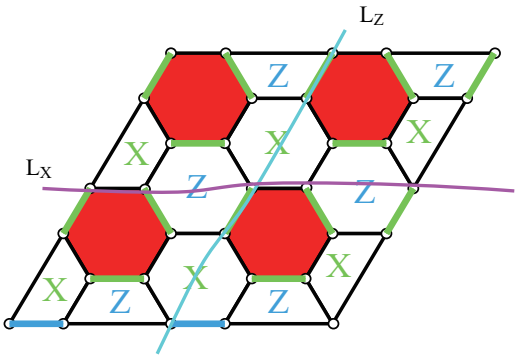

Fig. 11

また、エラー伝播により実効的な code distance が小さくなるのを防ぐこともできそうである。基本的に正方形を敷き詰めて作る surface code と考え方は同じである。Fig.11 でいうと、6-weight の多体 pauli 測定をする際に ancilla qubit からのエラーについて、水平方向に X エラーが、水平から 60° の方向に Z エラーが蓄積しなければいい。それを満たす手順とエラー伝播の種類を Fig.12 に示す。

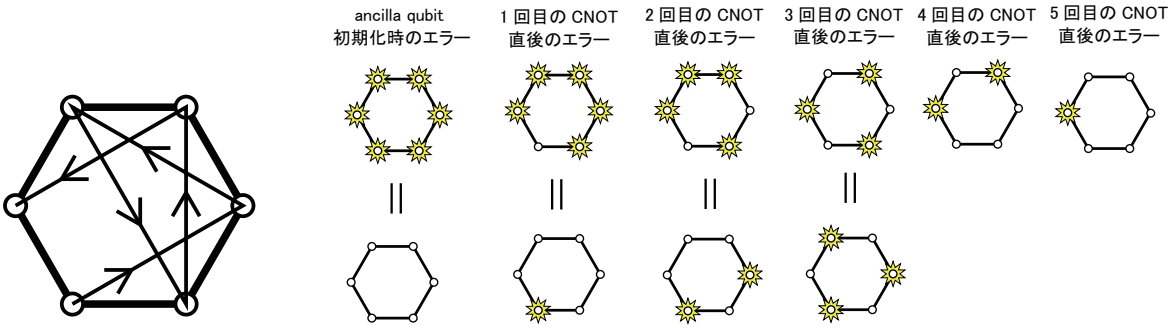

Fig. 12

Fig.12 から、水平方向、水平から 60° 方向のどちらにも 2 つのエラーが伝搬していないから、X,Z stabilizer の両方をこの CNOT 順序で実行すれば良い。

5. 拡大

最後に unfolding 操作と同時に拡大する手順を載せておく (Fig.13)。正方形 Surface Code の領域はオレンジが X stabilizer で青色が Z stabilizer である。

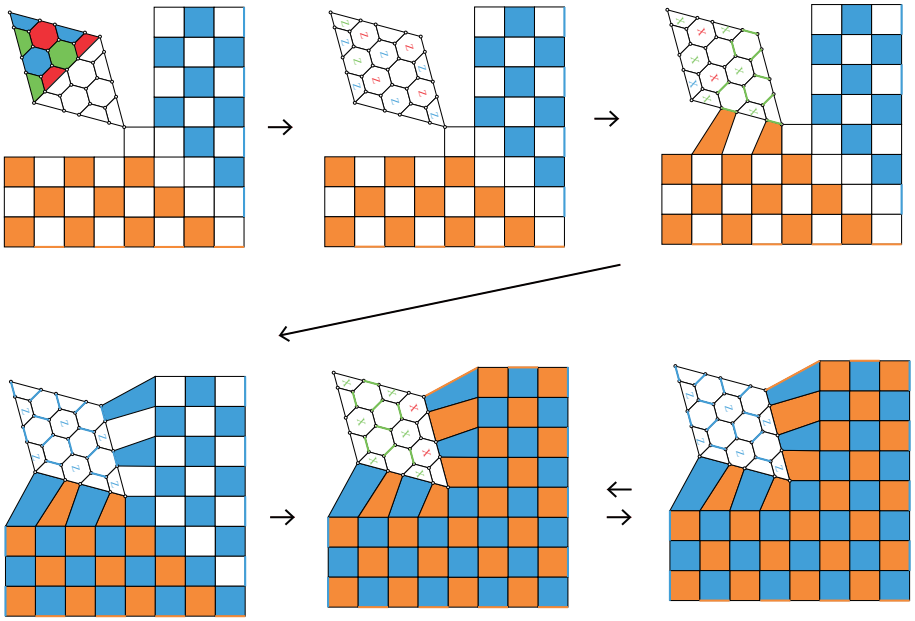

Fig. 13

おそらくこの手順で実行すればエラーを見つけながら unfolding と拡大を同時にできる。